Pokemon

Projet BA2 Informatique 12 mai 2013

LECOMTE Marceau, MOURAUX Antoine, VAN EECKHAUTE Mathieu

---

1 Introduction

Dans le cadre du projet du cours d'informatique pour BA2 en polytech, il a été demandé aux étudiants de choisir un sujet parmi une liste proposée. Nous avons choisi d'implémenter un jeu Pokemon, se rapprochant du vrai jeu.

Le but était pour nous de créer le jeu de manière à pouvoir ajouter du contenu facilement par la suite (de nouvelles cartes, des nouveaux pokemons, de nouveaux types d'objets, etc). Pour ce faire, la programmation orientée objet est un outil indispensable.

2 Mode d'emploi

2.1 Paramètres du jeu

Le jeu Pokemon est caractérisé par différents paramètres de jeu :

- Le joueur (son nom, ses pokémons et ses items)
- Les pokémons sauvages
- Les dresseurs (leur nom, leurs pokémons et leurs items)
- La carte de jeu (disposition des maisons, herbes hautes, dresseurs, arbres, plantes, etc)

Le jeu a été conçu de telle façon que le joueur puisse facilement modifier tous ces paramètres de jeu. Ainsi, s'il le désire, l'utilisateur a la possibilité de créer un jeu totalement personnalisé. Pour cela, il doit modifier les fichiers textes suivants :

2.1.1 Player

Le format de stockage est : nom% Pokemon1 ; niveau ; vie ; Attaque, dégats minimum, dégats maximum :AttaqueN, ... ! PokemonN %Item1type1 ; caractéristique ; charges !ItemNtype1 ; ...%Item1type2 ; caractéristique ; charges !ItemNtype2 ; ...

Voici un exemple :

> Sacha%Kangourex ;23 ;46 ;Coup de corne,35,43,5 !Pikachu ;7 ;35 ;Griffe,100,109,3 :Tonnerre,10,15,20 !Elektor ;
> nerre,10,15,15 !Dracaufeu ;20 ;70 ;Lance flamme,15,20,10 :Morsure,5,20,15%Baie rouge ;10 ;2 !Baie
> noire ;20 ;1%Pokeball ;40 ;2 !Superball ;60 ;1

2.1.2 Pokemon

Le format de stockage est : Pokemon ; niveau ; vie ; Attaque1, dégats minimum, dégats maximum : AttaqueN, ...

Miaous ;5 ;12 ;Griffe,3,9,15 :Leche,1,4,40

Mewtwo ;30 ;43 ;Laser,10,15,10

2.1.3 Dresseurs

Le format de stockage est : nom,% Pokemon1 ; niveau ; vie ; Attaque, dégats minimum, dégats maximum : AttaqueN, ... ! PokemonN ;... %Item1 ; caractéristique ; charges ! ItemN ; ...

MichMich%Miaous ;5 ;12 ;Griffe,3,9,15 :Leche,1,4,20 !Mewtwo ;12 ;25 ;Meteore,15,20,10 :Laser,1,3,40%Baie rouge ;10 ;2

Le roi des belges%Pikachu ;9 ;18 ;Griffe,3,9,15 :Eclair,1,4,40%Baie noire ;20 ;1

2.1.4 Map

Le format de stockage est : position x, position y ; type d'élément. Voici un exemple qui permet de placer un dresseur (7), une maison (1) et une plante (3). Il est possible de placer des plantes (5) et des arbres (6).

22,13 ;7

2,4 ;1

7,22 ;3

2.2 Déplacement

Lorsqu'il lance le jeu, l'utilisateur se trouve dans une carte. Il peut alors s'y déplacer. Le but du jeu est de battre tous les Pokemons présents dans l'herbe, ainsi que tous les dresseurs.

FIGURE 1 – Carte dans laquelle le joueur peut se déplacer

2.2.1 Herbes

En se déplaçant dans l'herbe, le joueur a une chance sur dix de tomber sur un Pokemon, et donc d'entrer en combat contre celui-ci (voir le mode d'emploi du duel plus bas). Il y a un nombre limité de Pokemons présents dans l'herbe, et lorsque le joueur en a battu un, il ne réapparaitra plus.

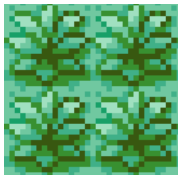

FIGURE 2 – Herbe

2.2.2 Dresseurs

Au détour d'une promenade, le joueur peut tomber face à un dresseur. Celui-ci possède un ou plusieurs Pokemons et des objets (il peut notamment soigner ses Pokemons). Une fois battu, un dresseur n'importunera plus le joueur.

FIGURE 3 – Dresseur

2.3 Duel

Un duel se déclenche dans 2 cas : quand le joueur se balade dans l'herbe ou quand il se trouve sur la case sous un dresseur. Lorsqu'un duel se déclenche, l'affichage de la carte laisse place au duel. Chaque situation donne lieu à un type de duel différent :

- quand le joueur est opposé à un pokémon dans l'herbe : Le joueur, en plus de combattre le pokemon, peut alors le capturer ou mettre un terme au duel en prenant la fuite.
- quand il est opposé à un dresseur : Le joueur n'a alors d'autre choix que de se battre, il ne peut ni fuir ni capturer le pokemon adverse.

2.3.1 Affichage

Les deux pokemons opposés sont représentés. Le pokemon du joueur est en bas de la fenêtre et le pokemon adverse en haut.

FIGURE 4 – Duel

Pour chaque pokemon, l'affichage comprend :

- une image le représentant (encadré rouge Figure 4)
- un encadré reprenant le nom du pokemon, ses points de vie, son niveau, son expérience (pour le pokemon du joueur), le nombre de pokemons restant à l'adversaire (pour le pokemon adverse) (encadré noir Figure 4)
- une boîte de dialogue en bas de la fenêtre avec une ligne de séparation. Au dessus et sous la ligne de séparation, on retrouve respectivement l'action du pokemon du joueur et l'action du pokemon adverse. (encadré vert Figure 4)

2.3.2 Le menu

Une fois le duel affiché, l'utilisateur doit sélectionner une action à effectuer.
Il sélectionne son action à l'aide d'un menu situé en haut à gauche de la fenêtre de jeu. Ce menu est composé d'onglets comprenant chacun des items. Sélectionner un item effectue l'action.
Les 4 onglets sont :

- Attaques (Figure 5(a)) : il comprend les attaques que le pokemon du joueur peut effectuer sur

le pokemon adverse.

– Pokemons (Figure 5(b)) : comprend les items permettant au joueur de changer son pokemon courant

– Objets (Figure 6(a)) : permet d'utiliser un objet parmi les items sur le pokemon du joueur

– Fuite (Figure 6(b)) : Permet de prendre la fuite quand c'est possible.

Attaques Pokemons Objets Fuite
Utiliser l'attaque Griffe sur le pokemon Chenipan (Max 109PV) (x3)
Utiliser l'attaque Tonnerre sur le pokemon Chenipan (Max 15PV) (x20)
Chenipan 12 / 12
Level : 5 Nombre de pokemons : 1
Pikachu 30 / 35
Level : 7
EXP 0 / 50
Sacha utilise l'objet Baie rouge sur le pokemon Pikachu
Pokemon sauvage : attaque Spores sur Pikachu!
Attaques Pokemons Objets Fuite
Changer le pokemon Kangourex avec le pokemon Pikachu
Changer le pokemon Kangourex avec le pokemon Elektor
Changer le pokemon Kangourex avec le pokemon Dracaufeu
Ronflex 18 / 59
Level : 26 Nombre de pokemons : 1
Kangourex 44 / 46
Level : 23
EXP 0 / 50
Sacha : attaque Coup de corne sur Ronflex!
Pokemon sauvage : attaque Repos sur Kangourex!

FIGURE 5 – (a)Duel Menu Attaques (b)Menu Pokemons

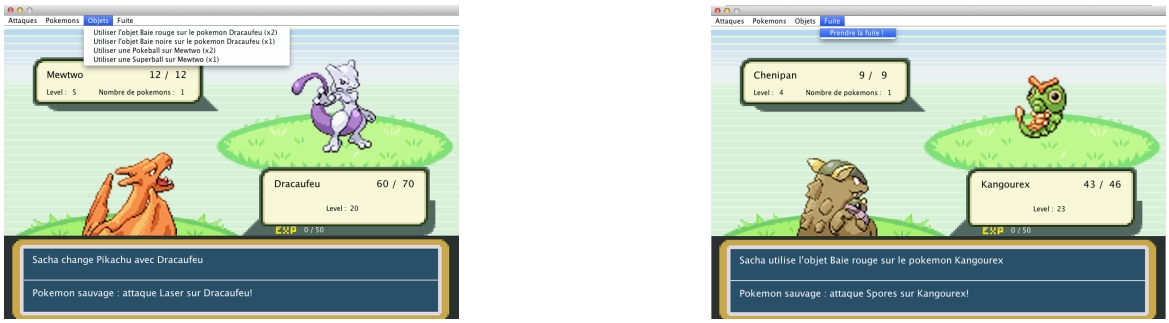
Attaques Pokemons Objets Fuite
Utiliser l'objet Baie rouge sur le pokemon Dracaufeu (x2)
Utiliser l'objet Baie noire sur le pokemon Dracaufeu (x1)
Utiliser une Pokeball sur Mewtwo (x2)
Utiliser une Superball sur Mewtwo (x1)
Mewtwo 12 / 12
Level : 5 Nombre de pokemons : 1
Dracaufeu 60 / 70
Level : 20
EXP 0 / 50
Sacha change Pikachu avec Dracaufeu
Pokemon sauvage : attaque Laser sur Dracaufeu!
Attaques Pokemons Objets Fuite
Prendre la fuite !
Chenipan 9 / 9
Level : 4 Nombre de pokemons : 1
Kangourex 43 / 46
Level : 23
EXP 0 / 50
Sacha utilise l'objet Baie rouge sur le pokemon Kangourex
Pokemon sauvage : attaque Spores sur Kangourex!

FIGURE 6 – (a)Menu Objets (b)Menu Fuite

3 Architecture

L'architecture du programme s'articule autour d'un Pattern MVC. Dans celui-ci, le modèle et la vue suivent un State Pattern : une classe s'occupe de "dispatcher" les informations, dépendant de la phase de jeu dans laquelle le joueur se trouve. (Carte ou duel)

3.1 Pattern MVC

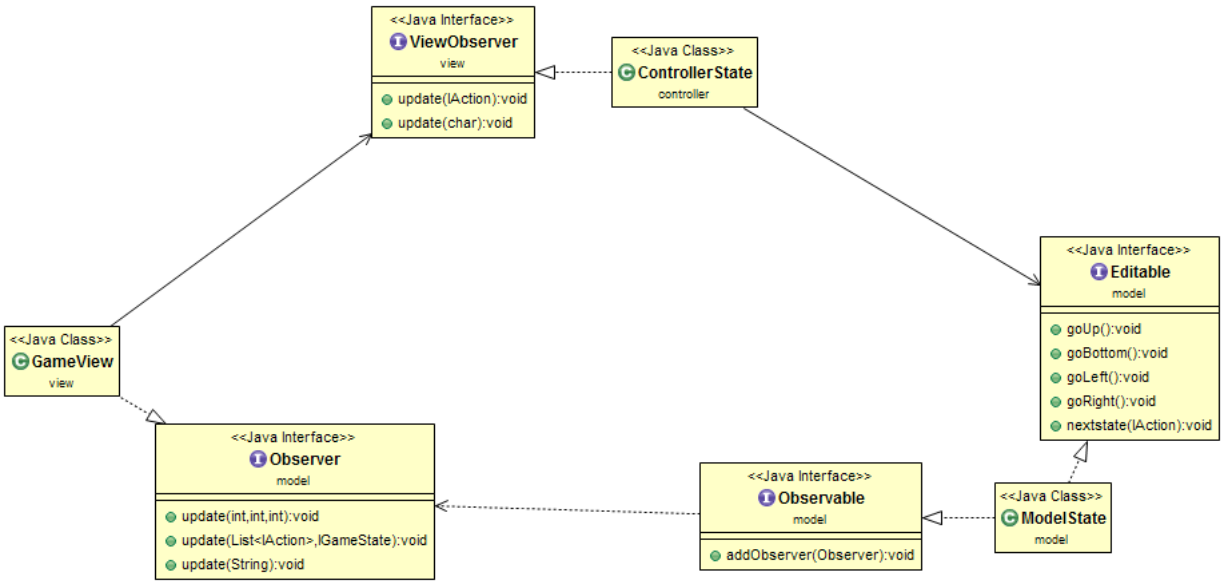

FIGURE 7 – Pattern MVC

Afin de comprendre pleinement l'architecture MVC implémentée ici, il est nécessaire d'expliquer le principe d'*Observable* et d'*Observer*.

3.1.1 *Observable* et *Observer*

Pour instaurer une communication qui va dans un sens unique entre un élément et un ensemble d'autres éléments, et afin d'éviter au maximum le couplage, on instaure des interfaces : *Observable* et *Observer*. On peut se représenter ce concept comme un cours ex cathedra : le professeur (*Observable*) parle aux élèves (*Observers*) sans que ceux-ci ne puissent intervenir. La communication va de l'*Observable* aux *Observers*. (voir figure [7], partie encadrée en rouge)

L'*Observable* ne connait absolument rien des *Observers* qui lui sont liés, à part les méthodes présentées dans l'interface *Observer*, qui sont des méthodes de mise à jour. Pour en revenir à notre cours, le professeur ne voit pas les élèves, et peut par exemple être filmé par une caméra. La seule chose qu'il fait, c'est parler (mettre à jour les *Observers*), peu importe qui écoute. Les *Observers*, quant à eux, sont mis à jour par l'*Observable* et utilisent l'information reçue à leur convenance.

Ceci dit, pour que les *Observers* puissent être mis à jour, l'*Observable* doit avoir une liste de tous les *Observers* qui l'écoutent.

3.1.2 Modèle, vue et contrôleur

Le concept de ce pattern est de pouvoir adapter différentes interfaces graphiques (vues) sans modifier le contenu du jeu (modèle). Ainsi, le modèle peut potentiellement communiquer avec différentes vues.

Modèle

Le modèle constitue le coeur du programme. C'est dans celui-ci que tous les calculs sont effectués. Dans notre configuration, le modèle implémente l'interface *Observable*.

Vue

La vue, quant à elle, est chargée de l'affichage. C'est l'interface graphique du programme. Chaque vue (dans notre cas, il n'y en a qu'une) implémente l'interface *Observer*.

Contrôleur

Enfin, le contrôleur est là pour faire le lien entre la vue et le modèle. C'est lui qui analyse les interactions avec l'utilisateur. Par exemple, si l'utilisateur pousse sur la lettre "z", le contrôleur va prévenir le modèle qu'il doit faire monter le personnage.

NB : Dans la configuration choisie, le contrôleur et la vue ne peuvent exister séparément. Dès lors, il n'est pas nécessaire de définir une interface du type *Observable* pour la vue, car elle ne possède de toute façon qu'un *Observer* et que le contrôleur n'envoie en aucun cas d'information à la vue.

3.2 Modèle

Dans le modèle, le *State Pattern* a été utilisé. Le *State Pattern* peut être vu comme un centre de dispatching qui, selon la phase du jeu (duel ou déplacement), oriente les messages reçus vers la partie du code qui en a besoin. Il s'agit de la classe ModelState qui possède deux états représentant respectivement la phase de déplacement (PositionModel) et la phase de combat (Duel).

3.2.1 Déplacement

Le déplacement est géré par deux classes. La classe PositionModel qui s'occupe des mouvements du joueur dans la map. La classe MapModel qui empêche le joueur de se déplacer quand c'est interdit (sur une maison, sur un arbre, ...) [1], elle va aussi dire au ModelState de créer un duel lorsque c'est

1. La position du joueur et des éléments de la carte est ici simplement une paire d'entiers(pos X, pos Y) convertie en un entier stocké dans un 'set' (liste triée) d'entiers. Les positions interdites ou spéciales (l'herbe ou la position des dresseurs) sont stockées dans des 'sets' d'entiers. On vérifie à chaque fois si la position courante du joueur est contenue dans ces 'sets' pour définir des interactions (entrer en combat ou bloquer le déplacement). Le lecteur intéressé remarquera

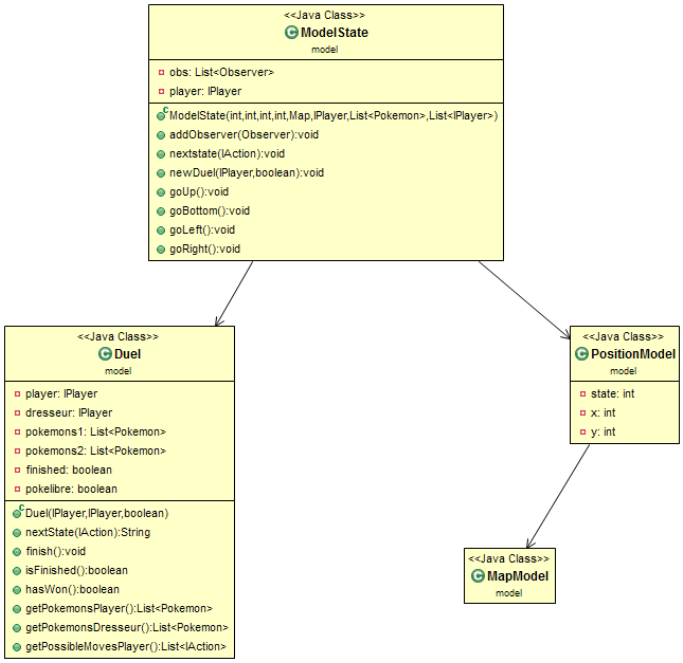

FIGURE 8 – Dispatching du ModelState

nécessaire.

3.2.2 Duel

Cette classe gère les combats entre dresseurs et joueur ou entre pokémons sauvages et joueur. Elle effectue un tour de jeu à l'aide de sa méthode nextState. Cette méthode reçoit une action, l'effectue, et effectue l'action de l'adversaire. Enfin, elle met à jour la liste des pokémons de chaque combattant, rafraîchit les actions possibles et termine le duel.

3.3 La Vue

La vue est responsable de l'interface graphique du jeu. Elle est composée d'une GameView qui implémente l'interface Observer du modèle. C'est donc un observer qui observe l'interface IGameState du modèle. Elle est responsable de créer et fermer la fenêtre de l'interface graphique ; gérer le changement d'affichage entre la map (conteneur) et le duel (géré par DuelView).

Il y a donc deux parties dans l'affichage : le duel et le déplacement dans la carte.

qu'il aurait été possible de créer une interface 'ICase' définissant les cases ainsi qu'une classe par type de case en fonction de l'interaction avec le joueur. Il a été décidé de ne pas adopter cette option car cela reviendrait à générer un objet par case de jeu. La création de l'objet case et les interactions en fonction de la position du joueur auraient été plus lourds en terme de gestion des ressources

3.3.1 L'affichage du déplacement

A une classe Conteneur (un JPanel), on ajoute l'image de la carte de jeu et du personnage. Le Conteneur affiche la partie visible de la carte en maintenant le personnage au centre. La carte est donc plus grande que sa partie affichée dans le conteneur. Il capte aussi les saisies clavier de l'utilisateur et en avertit le contrôleur.

Les images de la carte de jeu entière et du personnage sont respectivement générées par les classes MapView et Perso

3.3.2 L'affichage du duel

L'affichage du duel est géré par la classe DuelView. Elle est responsable de créer et mettre à jour le menu (Classe Menu) et le DuelPanel en fonction de l'évolution du duel.

Le menu du duel est géré par la classe Menu. Elle met à jour les items de chaque onglet du menu en fonction des actions possibles reçues et notifie le contrôleur avec l'action choisie par le joueur.

L'affichage des pokemons, de leurs caractéristiques et de la description des actions est pris en charge par la classe DuelPanel.

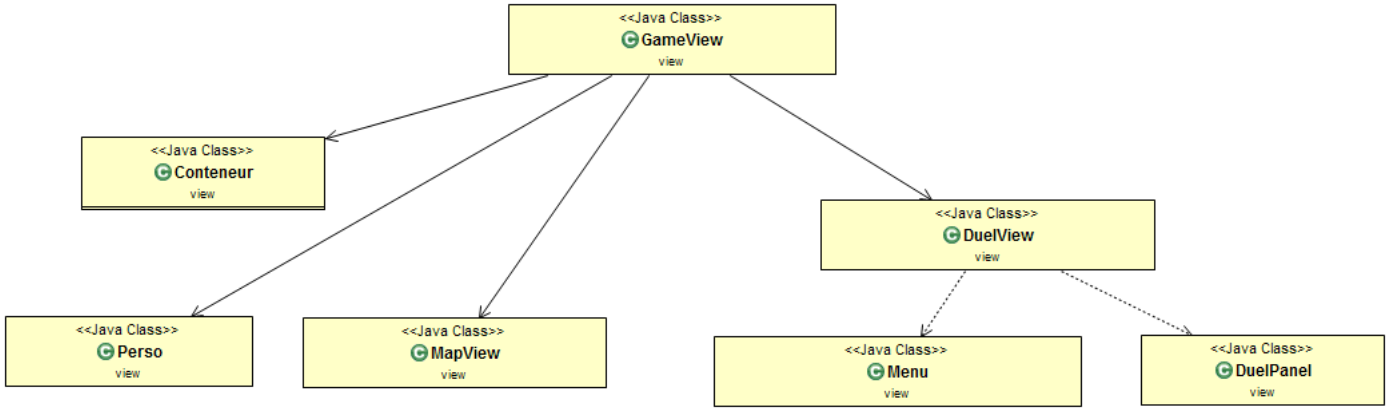

FIGURE 9 – Diagramme de classes Vue

3.4 Exemple de Cycle de jeu

3.4.1 Déplacement

Pour se déplacer, l'utilisateur enfonce une touche, le conteneur écoute le clavier et transmet la touche enfoncée au contrôleur. Celui-ci traduit la touche reçue en une action et appelle une méthode de déplacement du ModelState. Celui-ci appelle alors une méthode de changement de position de PositionModel, qui vérifie avec le MapModel si le déplacement est possible.

Une fois la position mise à jour, le ModelState peut envoyer cette nouvelle position à ses observeurs. (Dans ce cas, GameView) Ce dernier met alors à jour son interface graphique, et attend la prochaine instruction de l'utilisateur.

A l'étape de vérification de la position dans MapModel, si l'utilisateur doit rentrer en combat (avec un pokemon sauvage ou un dresseur), MapModel signale au ModelState de créer un nouveau duel.

3.4.2 Duel

Dans un duel, l'utilisateur choisit une action dans une barre de menu. L'action est transmise au contrôleur. Celui-ci appelle une méthode du ModelState qui appelle lui même la méthode nextState de Duel. NextState va gérer un tour de jeu du duel, c'est-à-dire : effectuer l'action du joueur, effectuer l'action de l'adversaire et mettre fin au duel si nécessaire.

Une fois le combat mis à jour, le ModelState va envoyer l'état du duel et la liste des actions possibles pour le joueur à ses *Observers* (GameView) Ce dernier met son interface graphique à jour et attend la prochaine instruction de l'utilisateur.

4 Améliorations possibles

La voie la plus directe pour améliorer le jeu est simplement d'ajouter du contenu : ajouter des pokemons, des dresseurs, agrandir la carte.

Avec des modifications mineures, on peut apporter d'autres améliorations :

- Ajouter des cartes : Le Card Layout présent dans la vue permet de changer facilement de vue. En ajoutant une surcharge de l'update de la vue, on peut arriver à passer en argument une Map. Le fait de passer d'une carte à l'autre serait alors géré par le MapModel.
- Rentrer dans les maisons : la détection des tentatives pour rentrer dans une maison existe déjà. En reprenant le point d'au-dessus (ajout de cartes), il suffit alors d'ajouter une carte qui correspond à l'intérieur d'une maison.

Enfin, d'autres améliorations sont envisageables, moyennant des modifications moins directes du code : enregistrer la partie pour pouvoir la recharger plus tard, ajouter un magasin pour acheter des objets, etc.